Introducing Microsoft DotNet (Article)

Author: Christophe Lauer
Category: VB.NET/ASP.NET
Difficulty: Intermediate

Version Compatibility: *Visual Basic.NET ASP.NET*

This code has been viewed **10683** times.

Introducing Microsoft DotNet

Reprinted with permission of Techmetrix Research (http://web.archive.org/web/20020702162429/http://www.techmetrix.com/

Introduction

At the Professional Developers' Conference in Orlando last July, Microsoft unveiled its latest architecture, .NET. Its various features and components were explained to the large audience by a number of speakers.

So what is .NET?The term is, essentially, a new marketing label which Microsoft is sticking on existing and future products. The .NET label now features on server products such as BizTalk Server 2000 and Application Center 2000, which are based on Windows DNA 2000 technology. The most interesting feature of .NET, however, lies in the development platform, languages and protocols which it emphasizes.

By bringing us .NET, Microsoft is presenting us with a new platform designed to facilitate development of interoperable Web applications, based on a totally new architecture. For Microsoft, .NET will be a way of "programming the Web," no less. Today, the first versions of Visual Studio .NET are available, and they enable us to sketch out a relatively accurate profile of how the .NET platform is likely to look in the long run.

Aims and objectives

The goal that Microsoft has set itself is ambitious, to say the least, both in technical and strategic terms. The new .NET platform has not evolved from the DNA 2000 technology currently available; rather, it is a totally new technology which is likely to shake up more than a few deep-rooted ideas.

.NET is an entirely new platform and technology which introduces a host of new products, whose compatibility with existing technology is not always guaranteed. It offers support for 27 programming languages, which share a hierarchy of classes providing basic services. .NET applications no longer run in native machine code, having abandaned Intel x86 code in favor of an intermediate language called MSIL which runs in a sort of virtual machine called the Common Language Runtime (CLR).

In addition, .NET makes intensive use of XML, and places a lot of emphasis on the SOAP protocol. Thanks to SOAP, Microsoft is hoping to bring us into a new era of programming which, rather than relying on the assembly of components or objects, is based on the reuse of services. SOAP and Web Services are the cornerstones of the .NET platform.

However, there is no need to start worrying yet about the future of DNA applications currently in production; as Microsoft themselves have admitted, the final version of .NET will not be available until early 2002, and .NET is able to run existing applications in native mode, without giving them all the .NET benefits.

Contrary to what Microsoft would have us believe (apparently in an aim to reassure current customers), changes run very deep and affect almost every component in the Microsoft DNA architecture:

- The IIS Web Server has dropped its effective but fragile multi-threaded model in favor of a multi-process model reminiscent of the Apache model.
- ASP technology gives way to ASP.NET (initially called ASP+), where interpreted scripts are replaced by codes compiled when they are first invoked, as for JSPs.
- Win32 APIs such as ATL and MFC are replaced by a coherent set of Base Framework classes.
- VB.NET no longer ensures ascending compatibility from VB6, as this language receives a lot of contributions (inheritance, .) in order to comply with the Common Language Specification (CLS) agreement.
- COM+ 2.0 is a totally original distributed components model which does not retain any inherited element from the COM/DCOM/COM+ lineup. To this end, COM+ 2.0 no longer uses the Windows Registry to register local or remote components: deployment of components in .NET will take you back to the good old days when installing a program meant copying files into a directory and uninstalling involved nothing more complicated than deleting the files.
- A new programming language called C# ("C sharp") is born: this is a modern object-oriented language, something of a cross between C++ and Java . C# was created by Anders Hejlsberg, architect of a number of languages and tools at Borland, including the famous Delphi.
- The new programming model, based on SOAP and Web Services, fundamentally changes the way in which applications are designed, and opens the way for a new profession: online provision of Web services.

These changes are moving towards a looser coupling between the Windows 2000 operating system and upper layers offering application server services. We look at these changes in more detail below, so as to give you an insight into the transformations that are taking place.

What is more, these technical changes, linked to the fact that the .NET platform will be a massive user of standards from independent bodies such as the W3C, the IETF and the ECMA, are leading a lot of analysts (including the Gartner Group) to surmise that "Microsoft is opening up."

From a strategic point of view, Microsoft has found a way to occupy a position of predominance on the Internet; the company has been out to achieve this for a long time, but until now, it had not found the means to do so. (We remember the episode in which Internet Explorer was provided free of charge, ready-installed on every PC equipped with Windows; Internet Explorer's many proprietary functionalities were extremely detrimental to rival Netscape.)

Today, with .NET, Microsoft is sending us a vision of an Internet made up of an infinite number of interoperable Web applications which together form a global service exchange network. These Web Services are based on the Simple Object Access Procotol (SOAP) and XML. SOAP was initially submitted to the IETF by DevelopMentor, Microsoft and Userland Software. Today, a number of vendors, including IBM, are greatly involved in SOAP.

Not only are these Web Services likely to develop on the Internet, but they may also change the way we plan information systems in enterprises, by making SOAP systematically used as application integration middleware, playing the role of a simple but efficient, standard EAI. An enterprise information system could then also become a network of front and back-office applications which interoperate through SOAP, and reciprocally use the Web Services that they implement.

It is not rash to suppose that Microsoft, through the numerous stakes it has acquired in multimedia content publishers, will soon become a provider itself, by hiring out or offering subscription to numerous Web services.

In the meantime, however, other vendors are not sitting back: IBM and, more recently, Oracle have announced offerings which enable the creation of Web Services. IBM, which has long been a supporter of SOAP, offers its "Web Services Development Environment" on its Alphaworks site, while Oracle has also just adopted SOAP, within 9i. Oracle has dubbed its offering "Dynamic Services", but it does not seem to be clearly defined as yet.

Therefore, Web Services will help Microsoft move from a model in which the majority of its revenues come from sales of boxed products and licenses designed for individual micro-computers, to a model revolving around subscription and hire of services carried by software infrastructures, parts of which, we can assume, will be provided free of charge. A few coinciding rumors suggest that Microsoft may eventually distribute its SDK and command line compilers free of charge, and would only market Visual Studio .NET. This is the sort of strategy that was behind the success of Java, where JDKs have been provided free by Sun since the outset.

This is a rather daring change in business strategy. Financial analysts agree that such changes of direction only rarely enable market leaders to keep their position at the top. The example of IBM, which never managed to regain the position it occupied when central computers reigned supreme, speaks volumes.

Some believe that this change in strategy is a clever move in more ways than one, as it will enable a number of lawsuits brought by the US Department of Justice to be nipped in the bud, and may allow Microsoft to smoothly ride the wave of change which lies ahead, under pressure from users who want to use IT resources and, above all, Internet from different mobile devices, not just from their home or office PCs.

Unpacking the .NET architecture

What exactly do we mean by .NET? In Microsoft marketing speak, all forthcoming versions of desktop and server software will carry the ".NET" label; this will be the case for the Office suite, the SQL Server database, and the Biztalk Server.

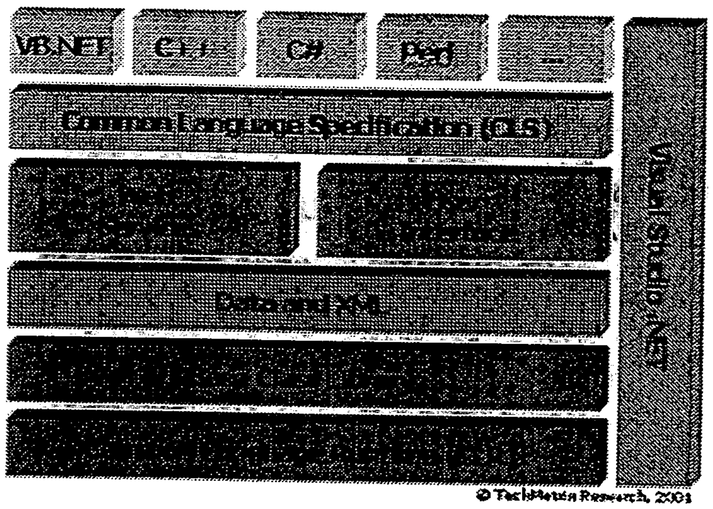
We will be looking in most detail at the architecture components and tools used to design and create enterprise Web applications.

With this in mind, we can describe the .NET architecture as follows:

- It is a set of common services which can be used from a number of object languages.
- These services are executed in the form of intermediate code that is independent of the underlying architecture.
- They operate in a runtime (Common Language Runtime) which manages resources and monitors application execution.

On reading this short description, one can be forgiven for drawing a parallel with Java. Indeed, Microsoft makes no secret of the fact that it drew its inspiration, in the most pragmatic sense, from existing technology and Java in particular.

The primary goal of .NET is to provide developers with the means to create interoperable applications using "Web Services" from any sort of terminal, be it a PC, PDA, mobile phone, and so forth.


VB.NET
C++
C#
Perl
...
Common Language Specification (CLS)
Data and XML
Visual Studio.NET
© TechMetrix Research, 2001

General .NET Architecture

.NET is multi-language
With the .NET platform, Microsoft will provide several languages and the associated compilers, such as C++, JScript, VB.NET (alias VB 7) and C#, a new language which emerged with .NET.

Third party vendors working in partnership with Microsoft are currently developing compilers for a broad range of other languages, including Cobol, Eiffel, CAML, Lisp, Python and Smalltalk. Rational, vendor of the famous UML tool Rose, is also understood to be finalizing a Java compiler for .NET.

Applications are hardware-independent
All these languages are compiled via an intermediate binary code, which is independent of hardware and operating systems. This language is MSIL: Microsoft Intermediate Language. MSIL is then executed in the Common Language Runtime (CLR), which basically fulfills the same role as the JVM in the Java platform. MSIL is then translated into machine code by a Just in Time (JiT) compiler.

Applications are portable
Applications compiled as intermediate code are presented as Portable Executables (PEs). Microsoft will thereby be able to offer full or partial implementations of the .NET platform over a vast range of hardware and software architectures: Intel PCs with Windows 9x, Windows NT4, Windows 2000 or future 64 bit Windows versions, microcontroller-based PDAs with PocketPC (e.g. Windows CE),

and other operating systems too, no doubt.

All languages must comply with a common agreement

Computer languages are numerous. Traditionally, new languages have been created to respond to new needs, such as resolving scientific problems, making calculations for research, or meeting strong needs in terms of application reliability and security. The result is that existing languages are heterogeneous: some are procedural, others object-oriented, some authorize use of optional parameters or a variable number of parameters, some authorize operator overload, others do not, and so it goes on.

For a language to be eligible for the range of languages supported by the .NET platform, it must provide a set of possibilities and constructions listed in an agreement called the Common Language Specification, or CLS. To add a language to .NET, all that is required in theory is for it to meet the requirements of the CLS, and for someone to develop a compiler from this language into MSIL.

This seems fairly innocuous at first glance, but the restrictions imposed by CLS-compliance on the different .NET languages mean that, for example, Visual Basic .NET ends up becoming a new language which retains little more than the syntax of Visual Basic 6.

The fact that all the .NET languages are compiled in the form of an intermediate code also means that a class written in a language may be derived in another language, and it is possible to instantiate in one language an object of a class written in another language.

Today, if you want to create a COM+ object, you generally have the choice between VB6 and Visual C++. But VB6 does not give access to all possibilities, and for certain requirements, you are restricted to VC++. With .NET, all languages will offer the same possibilities and generally offer the same performance levels, which means you can choose between VB.NET and C# depending on your programming habits and preferences, and are no longer restricted by implementation constraints.

At this point, you may be wondering how this can all be possible. Magic? Not really. In our opinion, there is no magic wand being waved here. To give a more even view of the multi-language aspect of .NET, we would prefer to say that .NET only supports one language, MSIL. Although Microsoft does let you choose whether to write this MSIL code using Visual Basic syntax, or C++ syntax, or Eiffel.

To put it frankly, in order to be able to provide the same services from languages as remote as Cobol or C#, you have to make sure these languages have a common denominator which complies with the demands of .NET. This means that the .NET version of Cobol has had to receive so many new concepts and additions that it has practically nothing left in common with the original Cobol. This applies just as much to the other languages offered in .NET, such as C++, VB, Perl or Smalltalk.

So what we need to understand is that when Microsoft announces the availability of 27 languages, we should interpret that as meaning there are 27 different syntaxes.

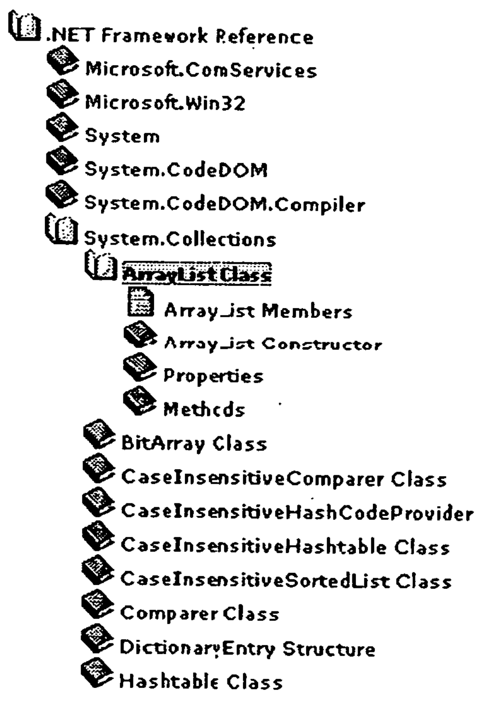
The most symptomatic example concerns Java. It is one of the intended .NET languages, thanks to Rational, who are currently working on a Java to MSIL compiler. But what kind of Java are we talking about? It is a Java which runs as MSIL code, not byte-code. This Java does not benefit from the traditional APIs offered by the J2EE platform, such as JMS, RMI, JDBC, JSP. This is a Java in which EJBs are replaced by .NET's distributed object model. The label says Java, the syntax says Java. but Java it ain't!

Of course, the case of Java is a bit of an exception. Indeed, Java specialists see .NET overall as a rather pale copy of Java itself, and consider it to be proof of Microsoft's successive attempts to undermine Java's future. Relations between Sun and Microsoft have been peppered with disputes and lawsuits in recent years. It was out of the question that Microsoft would participate in the construction of Java by offering total support for the language in its new .NET platform.

From our perspective, the support for Java in .NET is, as it stands, totally unusable if one intends to maintain a degree of compatibility with Sun's J2EE platform. We believe that its only justification is that it reinforces Microsoft's campaign to seduce developers. Microsoft's strategy is to win over developers in order to benefit from their prescriptive powers, with the aim of eventually imposing .NET on a wide scale.

Similarly, Microsoft is cleverly entertaining certain rumors, such as the recurring whispers predicting the eventual availability of .NET on Unix systems, even Linux. Linux is increasingly popular among developers, and is becoming a potential alternative to Windows NT as far as server architectures are concerned. By keeping details hazy around the issue of Linux support for .NET, Microsoft can win over fans of the free operating system.

All the languages use a coherent set of basic services
A hierarchical set of classes provides all the services and APIs necessary for application development. Thanks to the introspection capabilities offered by the reflection API, code is self-documented, which gives the developer exhaustive documentation, as with Javadoc.



```
.NET Framework Reference
    Microsoft.ComServices
    Microsoft.Win32
    System
    System.CodeDOM
    System.CodeDOM.Compiler
    System.Collections
        ArrayList Class
            ArrayList Members
            ArrayList Constructor
            Properties
            Methods
        BitArray Class
        CaseInsensitiveComparer Class
        CaseInsensitiveHashCodeProvider
        CaseInsensitiveHashtable Class
        CaseInsensitiveSortedList Class
        Comparer Class
        DictionaryEntry Structure
        Hashtable Class
```

Example of Base Framework Class Hierarchy

Close-up on the CLR

Common Language Runtime

As has been mentioned already, the CLR is, like the Java virtual machine, a runtime environment that takes charge of resource management tasks (memory allocation and garbage collection) and ensures the necessary abstraction between the application and the underlying operating system.

In order to provide a stable platform, with the aim of reaching the level of reliability required by the transactional applications of e-business, the CLR also fulfills related tasks such as monitoring of program execution. In DotNet-speak, this involves "managed" code for the programs running under CLR monitoring, and "unmanaged" code for applications or components which run in native mode, outside the CLR.

The CLR watches for the traditional programming errors that for years have been at the root of the majority of software faults: access to elements of an array outside limits, access to non-allocated memory zones, memory overwritten due to exceeded sizes.

This monitoring of the execution of managed codes comes at a price, however. Although it is currently impossible, from performances returned by current Beta-test versions, to quantify the overhead incurred by application monitoring, we can expect performance to slip by at least 10%, as Microsoft admits. Of course, we might ask whether a 10% reduction in performance is such a bad thing if it leads to new levels of reliability and availability...

As Moore's law is always borne out when it comes to increasing processor performance, how long must we wait before we have servers which are 10% more powerful?

Close-up on ASP.NET

The new technology for creating dynamic Web pages is a full rewrite, based on the services of the CLR. To this end, any of the languages offered by .NET will be usable in the ASP.NET pages. Pages bear different extensions to ASP 3.0. So, simple pages take extension '.aspx', while Web Services take '.asmx' (for Assembly). Pagelets, which are a sort of reusable portion of ASP.NET pages, take extension '.aspc'.

A .NET application can therefore happily allow old .asp and .aspx pages to cohabit. Simply put, old-style ASP pages will run via the DLL asp.dll and will not benefit from the CLR's functionalities.

Now, .aspx pages are no longer interpreted but are compiled as MSIL code when first invoked, then run from the intermediate code produced, following the example of JSPs in the J2EE world. The logical result should be an improvement in performance, which Microsoft announces as comparable to that enjoyed by Visual Basic applications when moving to compiled version 5 from version 4.

These deep-rooted changes will entail a number of modifications to the way an ASP page is coded, in addition to the changes caused by the introduction of new concepts in VB.NET. It might be a good idea to become familiar with these modifications now, so as to be prepared for the next ASP applications written in VBScript and minimize the work required to migrate them to ASP.NET. We shall look at the main changes in order to give you an idea of the effort required to migrate from ASP to ASP.NET.

These changes will take place on three levels:

- Changes in the API
- Changes to page structure
- Changes between VBScript and VB.NET

ASP.NET only supports one type of language per page. In DNA, an ASP page could contain alternate sections of JScript and VBScript. In ASP.NET, this would be impossible, as one page leads to the creation of an MSIL code file after compilation.

In ASP.NET, functions must be flanked by HTML <SCRIPT> tags, and it would be impossible to explode an HTML-generating function into different bits. So, you will no longer be able to write:

```
<%
Function SayHello()
%>
<b><i>
<%
Response.Write " Hello ! "
%>
</i></b>
<%
End Function
%>
```

Instead, you will need to write:

```
<SCRIPT LANGUAGE="VB" runat=server>
Function SayHello()
Response.Write ("<b><i> ")
Response.Write (" Hello ! ")
Response.Write ("</i></b> ")
End Function
</SCRIPT>
```

The parentheses around the function-calling parameters are now mandatory. Other compatibility problems will come from the fact that all ASP.NET arrays are 0 based, while in ASP 3, some are 1 based, while others are 0 based.

In VB.NET, parameters will, by default, be passed by value (ByVal); currently, in VBScript, they are by default passed by reference (ByRef). Lastly, VB.NET will no longer support the default values, or keywords Set and Let.

Although these are not critical modifications, they will cost a lot of time, since existing codes will have to be changed if you want to be able to benefit from the capacities of the CLR and the compiled codes. Microsoft is already announcing that migration tools will be provided when the .NET platform is distributed, but there is no harm in getting into good habits.

As for COM components, ASP.NET will be able to make old COM components work by encapsulating them, but they will run outside the managed environment of the CLR, and the context changes between managed and non-managed will be costly in terms of performance. It is therefore highly likely that you will eventually decide to rewrite COM components as COM+ 2.0 components or Assemblies.

The argument that could clinch your decision to use ASP.NET is the introduction of controls on the server side. With these controls, your ASP.NET pages will benefit from visual or non-visual components that provide advanced services: TreeView, ListBox, Calendar, and so on. All these components analyze the type of Web client calling them, and generate a suitable representation. Typically, an entry field will use the client scripting functionalities of Netscape or Internet Explorer (JavaScript or DHTML) to validate the entry, but will validate on the server side for browsers where Javascript is not supported or has been deactivated.

.NET in the short term

The final versions of the full set of .NET components (whether this concerns the development tools, or products from the .NET server family) are unlikely to be available before mid -2001, going by the most optimistic predictions.

"Old" applications built on the Microsoft DNA architecture will still function on Windows 2000 servers equipped with .NET generation tools. The two generations of applications will be able to cohabit without interference.

We therefore do not see any short-term threat for current and future DNA-based applications.

Microsoft points out that tools and assistants to help with migration will be provided with the .NET platform. However, we do not feel that this is an ideal solution, for various reasons. Firstly, migration assistants can never carry out 100% of the modifications necessary. Consequently, it would be advisable to devote sufficient time and resources to this migration. Secondly,transforming an ASP/VBScript application into ASP.NET/VB.NET will not automatically make it a.NET application. It will in all likelihood be necessary to alter the application architecture, so as to benefit fully from the new possibilities offered by .NET.

In future articles we will try to answer some of the questions that you are undoubtedly asking yourself if you have Microsoft DNA applications in production. We will try and draw an accurate schema of the optimum .NET architecture, and will show you the best way to write DNA applications which can be ported to .NET.

.NET in the long term

Whether or not Microsoft achieves what it has set out to achieve with .NET, it cannot be denied that the way we design applications is going to undergo some changes. With the advent of e-commerce and B2B exchanges, there is already a need for interconnected applications which communicate via an enterprise network or through the Internet.

With this in mind, we can see that with .NET, Microsoft's main aim is to supply tools which can be used to develop applications as easily as Visual Basic did a few years ago, during the golden age of the client-server application.

What's the verdict?

Pragmatically speaking, and casting aside any preconceived ideas about the Redmond vendor, a clever strategy would be to carry out sustained technology tracking of .NET and as its alternatives, together with the technologies on which all of these are based, i.e. XML and SOAP.

Until the final version of .NET appears, we will continue to keep you informed of technological and strategic developments, with further TrendMarkers articles on Microsoft's DotNet. Stay tuned!

Find out more

Web resources:
http://web.archive.org/web/20020702162429/http://www.devx.com/dotnet/
http://web.archive.org/web/20020702162429/http://www.dotnettoday.com/
http://web.archive.org/web/20020702162429/http://www.dotnetwire.com/
http://web.archive.org/web/20020702162429/http://www.gotdotnet.com/ (Microsoft's site)

Mailing list:
http://web.archive.org/web/20020702162429/http://discuss.develop.com/dotnet.html

Recommended Book:

Professional VB.NET